UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/741,502	12/19/2000	Theodore S. Hills	TAJ-0001	5730

23413 7590 05/06/2005

CANTOR COLBURN, LLP
55 GRIFFIN ROAD SOUTH
BLOOMFIELD, CT 06002

EXAMINER	
VU, TUAN A	

ART UNIT	PAPER NUMBER
2193	

DATE MAILED: 05/06/2005

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary	Application No.	Applicant(s)	
	09/741,502	HILLS, THEODORE S.	
	Examiner	Art Unit	
	Tuan A. Vu	2193	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on 22 December 2004.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) 49-72 is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) 49-72 is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application (PTO-152)
6) ☐ Other: _____.

DETAILED ACTION

1. This action is responsive to the Applicant's response filed 12/22/2004.

As indicated in Applicant's response, claims 1-48 have been canceled and new claims 49-72 have been added. Claims 49-72 are pending in the office action.

Claim Objections

2. Claims 56, 64, and 72 are objected to because of the following informalities:

The above claims recite 'abase'(line 8, 10, 10, respectively), whereas the term 'abase' should be 'a base'; and the rejection is interpreting this term in light of the suggested correction.

Appropriate correction is required.

Claim Rejections - 35 USC § 103

3. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

4. Claims 49-54, 57-62, and 65-70 are rejected under 35 U.S.C. 103(a) as being unpatentable over Veldhuizen, USPN: 5,835,771 (hereinafter Veldhuizen), in view of Tanaka,USPN: 6,738,966 (hereinafter Tanaka); and further in view of Killian et al., USPN: 6,477,683 (hereinafter Killian)

As per claim 49, Veldhuizen discloses a method of compiling a source code written in an object-oriented source language into executable machine-language code for a target computer, comprising:

defining in said source code, using said source language, software-visible objects of said target computer as pre-existing objects of classes defined in said source language (e.g. col. 5, lines 29-37; template class - col. 8, lines 1-61; tables 1-4, cols. 11, 12), wherein at least one of member methods of said classes contain statements in said language, said statements containing expressions evaluating to values of operation codes and associated operand specifiers in said machine-language (e.g. col. 8, lines 1-61 - Note: inline instruction in template definition including operand specifiers (e.g. + > <=) reads on associated operand specifiers when template class are being defined and subsequently evaluating to values of operation codes and associated operand specifiers at link time);

But Veldhuizen does not explicitly disclose compiling results of said defining into said machine-language code by expanding inline, every invocation of said member methods of said classes to said values of operation code and associated operand specifiers to which their expressions evaluate; said expanding inline of invocations continuing recursively until all invocations of said member methods have been replaced by operation codes and associated operand specifiers, said compiling accomplished without requiring application of an intermediate language. The concept of providing inline directives at compiling time using source programming language similar to the target code as taught by Veldhuizen's compilation process implies the inline instructions when creating the object code prior to runtime, the inherent linking process will resolve those instructions and associated operation code and specifiers and iteratively provide for their replacement with appropriate link time values without intermediate language. Hence the limitation is disclosed.

Nor does Veldhuizen disclose that the software visible objects are hardware objects of computer. The use of assembler inline instruction to address resources usage/optimization issues in regard to computer hardware like registers or memory was a known concept; and this is evidenced by Tanaka's assembler directives being inlined (see col. 5, lines 46-67; Fig. 4,7,8) and purported just like in Veldhuizen (see Velhuizen BACKGROUND) to obviate useless repetition or overlapping of memory usage via obviating redundant allocation of registers wherein variables are no longer needed. Furthering the use of inline expansion by Tanaka to address hardware resources Killian discloses inline expansion applied in a simulation/debugging environment for preparing optimized code to implement a hardware circuitry via HDL and C++ for representing a microprocessor design (col. 26, lines 46-61; col. 27, line 47-54). Based on the teachings by Velhuizen and the purpose to obviate computer hardware resources, it would have been obvious for one of ordinary skill in the art at the time the invention was made to implement the inline expansion by Veldhuizen with inline directives by Tanaka to optimize register (i.e. hardware of target computer) usage approach by Tanaka and further to use such inline optimizing technique to implement hardware components for microprocessor design as by Killian. The motivation would be obvious because it is well-known that limited hardware resources for a given architecture require time expediency improvement (see Killian: col. 2, lines 35-67) and whereas inline code obviate external compiler resources at runtime as intended by Velduizen (col. 4, lines 12-19), the use of inline as by Killian's approach enables microprocessor designer-level instructions to be immediately converted into operand or opcode at runtime (see Killian: col. 26, line 46 to col. 27, line 54); because the register saving approach

by inline instructions enable flexibility and reusability enabling speeding execution as envisaged by Tanaka (col. 3, lines 61-67).

As per claim 50, Veldhuizen discloses defining source code with subclass implementing a method for invoking out-of-line a subroutine defining a call method, wherein each subclass defines an instance holding said subroutine (e.g. tables 1-4, col. 11-12 – Note: class with member routine calling other subclass member methods as shown by above tables reads on the limitation and context switching by main program when encountering a inline group or metaprogram class reads on out-of-line invocation).

But Velduizen does not explicitly disclose compiling said methods thus defined; compiling such compiling results by expanding one invocation of said methods of said subclass to be inline expansions, said subroutine to be invoked out-of-line as an object of said subclass of a corresponding call member method. Based on the rationale as set forth in claim 49, the compiling and linking resolving the internal representation of the source code inline group including specifiers and instructions code reads on the above limitation.

Nor does Veldhuizen explicitly disclose a subroutine operable for saving registers and restoring registers after the out-of-line call. The inline instructions with class objects (template class) calling member method leading to other subclass methods is disclosed by Veldhuizen (see tables 1-4, col. 11-12); and when an inline metaprogram class in encountered, the inherent saving and restoring registers prior to and after the invocation of said template class is disclosed.

As per claim 51, Velhuizen (in view of Tanaka and Killian) discloses hardware and software objects involved in subroutine being invoked out-of-line (see claim 50); objects being

allocated and deallocated by each subroutine (Note: saving and restoring registers state from claim 50 reads into this).

Velhuizen does not disclose evaluating use of hardware and software object across subroutine invocations based on results of said evaluating, code generating for allocating, deallocating, saving and restoring corresponding said hardware and software objects. But Velhuizen discloses recursively evaluating use of variables in a iteration and providing information for providing additional method calls addressing hardware and software objects invoked in the generating of source code having class and subclasses to be invoked as out-of-line (see col. 11, line 17 to col. 12, line 55); and Tanaka disclose recursive replacement of software/hardware objects to generate inline macro definition. Thus the evaluation based on a recursive process yielding previous evaluation is disclosed. Further, the out-of-line invocation and return therefrom as taught by Veldhuizen such that the results from such return are used by the main code inherently teach the automatic evaluating of resources being the results of Veldhuizen's implied allocating, deallocating, saving and restoring corresponding said hardware and software objects as shown in claim 49.

As per claim 52, Veldhuizen discloses source language to implement object-oriented data types; variables of said types; and compiling to instantiate objects representing said variables, said objects being instances of classes representing said object-oriented data types (e.g. tables 1-4, col. 11-12); but does not explicitly disclose that the OO data types are abstract data types (ADTs).

The concept of using abstract data type in design using object-oriented approach was well-known in design and meta-definition or modeling; and enhancing to the metaprograms by

Veldhuizen, Killian in a design approach for developing microprocessors uses C++ abstract data type (col. 20, lines 35-40) to describe a type of operation to simulate. Based on the rationale as to optimize hardware objects via software as set forth in claim 49, it would have been obvious for one of ordinary skill in the art at the time the invention was made to provide the ADT so well-known in OO modeling/design and programming language such as taught by Killian and use that in Veldhuizen's (combined with Tanaka/Killian) optimization endeavors in case there is a need to categorize a specific hardware type as taught by Killian.

As per claim 53, Veldhuizen discloses statements having expressions evaluating to values of op codes and associated specifiers (col. 8, lines 1-61 - Note: inline instruction in template definition including operand specifiers (e.g. + > <=) reads on associated operand specifiers when template class are being defined and subsequently evaluating to values of operation codes and associated operand specifiers at link time); but does not disclose first library for said class having said expressions; nor a second library for subclasses; nor third library for equivalent classes to the first library.

Tanaka teaches libraries enabling the reuse of inline routines so to alleviate syntax checking (e.g. col. 13, lines 9-13); and Veldhuizen already suggests a library usage to enhance efficiency subclass (e.g. col. 15, lines 15-17); and the object-oriented templates imply object-oriented polymorphism and variance of equivalent classes. It would have been obvious for one of ordinary skill in the art at the time the invention was made to provide a library to store the class and subclass as taught by Veldhuizen template metaprograms using Veldhuizen's library suggestion so the libraries as taught by Tanaka be used to store subclass or equivalent class can

be reused in the same manner intended by Tanaka to expedite execution without committing resources for verifying code being inserted.

As per claim 54, Veldhuizen discloses a method of compiling literals in a source code, written in an object-oriented source language, into a executable machine language code for a target computer, comprising:

defining in said source code, using said source language, classes having initializer member methods that take as arguments an internal representation of said literals of a compiler (e.g. col. 5, lines 29-37; template class - col. 8, lines 1-61; tables 1-4, cols. 11, 12 – Note: Template class methods declared in class and taking arguments as internal representation at compiler level reads on limitation); compiler performing at least one of: incorporating said initialized object into an output of said compiler; and using said initialized object in subsequent compilation steps, said initializer member methods to yield said initialized object (Note: compiling inline class template wherein class having arguments and resolving results of previously resolved objects during linking time reads on this limitation – refer to corresponding rationale of claim 49).

But Veldhuizen does not explicitly disclose that said initializer member methods are previously compiled from said source code; and upon parsing one of said literals said compiler calling one of said initializer member methods to yield said initialized object; the limitation reciting initializer member methods taking said internal representation of said parsed literal as an argument having been addressed above. Based on the linking process as mentioned above, the class template having methods (taking arguments being representation of source code literals) are submitted to a linking process wherein said precompiled methods are resolved, such process

parsing of such internal representation yielding a initialized object code ready for execution; and this is integral to every compiler to create an object code being a final machine level executable (refer to Veldhuizen's code being instantiated will result in code for running -- Fig. 2).

As per claim 57, this is a computer-readable medium version of claim 49; hence is rejected with the corresponding rejection as set forth therein.

As per claims 58-62, these are computer-readable medium versions of respectively claims 50-54; hence are rejected with the corresponding rejection as set forth therein, respectively.

As per claims 65-70, these are propagation medium versions of respectively claims 49-54; hence are rejected with the corresponding rejection as set forth therein, respectively.

5. Claims 55-56, 63-64 and 71-72 are rejected under 35 U.S.C. 103(a) as being unpatentable over Billday, "Class Hierarchy", October 13 2000, pgs. 1-2; whose URL is following: *<http://web.archive.org/web/20001013065435/http://www.billday.com/Classes/Docs/tree.html>*.

As per claim 55, Billday discloses a method of compiling two object-oriented classes from a single class literal in a source code, comprising:

in said source code, labeling at least one member method defined in said single class literal as belonging to a constant version of a class (e.g. *class java.lang.Object, java.awt.Component, java.lang.Number, java.lang.Exception* -pg 1-2) said constant version of said class differentiated from a variable version of said class (e.g. class java.applet.AppletContext; class java.applet.Applet, class java.lang.Integer, class.java.RuntimeException – pg. 1-2 – Note: a child class derived from a parent class and having differentiated methods as well as inheriting the characteristics of a parent class reads on

variable version of a parent class whose signature is a literal and unchangingly reused from deriving variable children classes or differentiated subclasses therefrom); and

generating two classes from said single class literal, said constant version of said class being a base class (e.g. java.lang.Exception, java.lang.RuntimeException – Note: one derived subclass *RuntimeException* from parent Exception reads on variable version from a base class whose signature is a constantly reused literal) of said variable version of said class.

But Billday does not explicitly disclose a compiler, however discloses a runtime class (Note: java *RuntimeException* reads on class being translated into executables). Based on such implied need to provide executable class objects, it would have been obvious for one of ordinary skill in the art at the time the invention was made to provide a compiler to convert Java classes into a runtime objects because classes being defined are to be converted before a runtime object can perform what it is supposed to do.

As per claim 56, Billday discloses a method of compiling two object-oriented class interfaces from a single class literal in a source code, comprising:

in said source code, labeling at least one member method defined in said single class interface literal as belonging to a constant version of a class interface (e.g. *class java.awt.LayoutManager, java.awt.image.ImageObserver, java.awt.MenuContainer* -pg 1-2) said constant version of said class interface differentiated from a variable version of said class interface (e.g. *java.awt.BorderLayout, java.awt.Component, java.awt.FlowLayout, java.awt.MenuBar* – pg. 1-2 – Note: a class having differentiated methods to implement a constant version of a signature or literal of an unimplemented parent class interface reads on

variable version of a constant class interface whose signature is a literal and unchangingly reused from deriving various versions of implementation classes, i.e. different instances of classes implementing a interface) ; and

generating two class interfaces from said single class interface literal, said constant version of said class interface being a base class of said variable version of said class interface (e.g *class java.awt.LayoutManager, java.awt.image.ImageObserver, java.awt.MenuContainer* - pg 1-2 – Note: any of the various class instances like *java.awt.BorderLayout, java.awt.Component, java.awt.FlowLayout, java.awt.MenuBar* for implementing a signature representing a base unimplemented class interface, e.g. LayoutManager, ImageObserver, MenuContainer, reads on variable version from a base class whose signature is a constantly reused literal).

The compiler limitation would have been obvious for the same reasons as set forth in claim 55.

As per claims 63-64, these are computer-readable medium versions of respectively claims 55-56; hence are rejected with the corresponding rejection as set forth therein, respectively.

As per claims 71-72, these are propagation medium versions of respectively claims 55-56; hence are rejected with the corresponding rejection as set forth therein, respectively.

Response to Arguments

6. Applicant's arguments filed 12/22/2004 have been fully considered but they are now moot in view of the new grounds of rejection being necessitated by the amendments.

Conclusion

7. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Tuan A Vu whose telephone number is (272) 272-3735. The examiner can normally be reached on 8AM-4:30PM/Mon-Fri.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Kakali Chaki can be reached on (571)272-3719.

The fax phone number for the organization where this application or proceeding is assigned is (571) 273-3735 (for non-official correspondence – please consult Examiner before using) or 703-872-9306 (for official correspondence) or redirected to customer service at 571-272-3609.

Any inquiry of a general nature or relating to the status of this application should be directed to the TC 2100 Group receptionist: 571-272-2100.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

VAT
April 24, 2005

TODD INGBERG
PRIMARY EXAMINER